NC STATE UNIVERSITY

Quantum Search with Wildcards

Alec Landow and Liam Adams

Motivation

- Pattern matching is an important component of DNA sequencing, search engines and AI.
- Classically the problem of finding the closest match of a pattern is O(MN), and exact match is O(M + N)
- Quantum computers can find a match without searching the entire input string
- We implemented an algorithm for finding the closest match of a pattern in a string, with wildcards
- This is useful in applications where data can mutate over time, an example of such data is DNA
- The quantum algorithm we implemented returns the starting index of the closest match for a pattern in a string in $O(\sqrt{N})$ iterations.

Algorithm Outline

- The algorithm consists of 3 main components: the initial state, oracles, and diffuser
- We create a quantum register for each character in the pattern.
- The state of the register represents the index of the character in the overall string.
- The pattern can only begin in N-M locations, so each register only needs to encode those positions
- “Compile Once, Run Many” Approach
- During each of the $O(\sqrt{N})$ iterations, a random character (register) from the pattern is selected, its oracle is applied, and diffusion is applied to amplify the state amplitude

Algorithm Outline

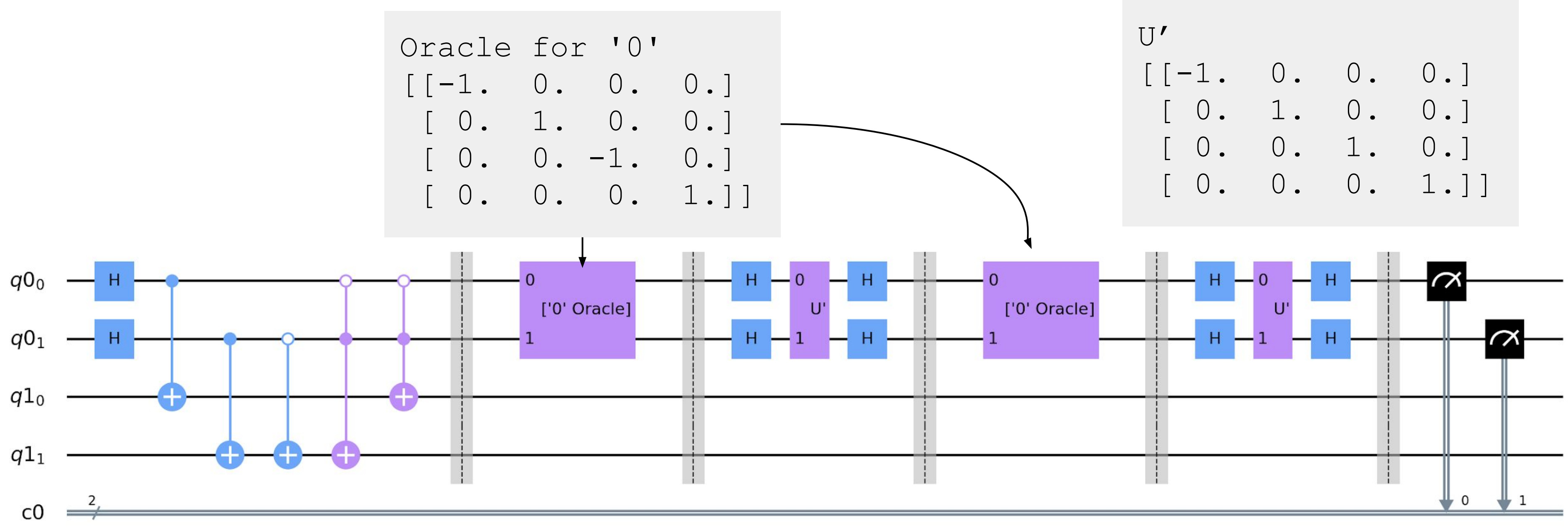
- Selecting a character from the pattern at random rather than passing the characters in order allows us to find the closest match
- The initial state is created so that the state of the second register is one position beyond the first and so on
- The oracles for the characters are diagonal matrices containing a -1 where that character appears in the string
- After the oracle is applied Grover diffusion is used to amplify the amplitude of the state
- Diffuser only applied to first register as that is the only state we are interested in measuring
- The state measured most frequently is the starting index of the pattern in the string

Circuit Examples

Input String: 0101

Pattern: 01

"Single Oracle", Matrix Diffuser



```
Oracle for '0'
[[-1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0. -1.  0.]
 [ 0.  0.  0.  1.]]
```

```
U’
[[-1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

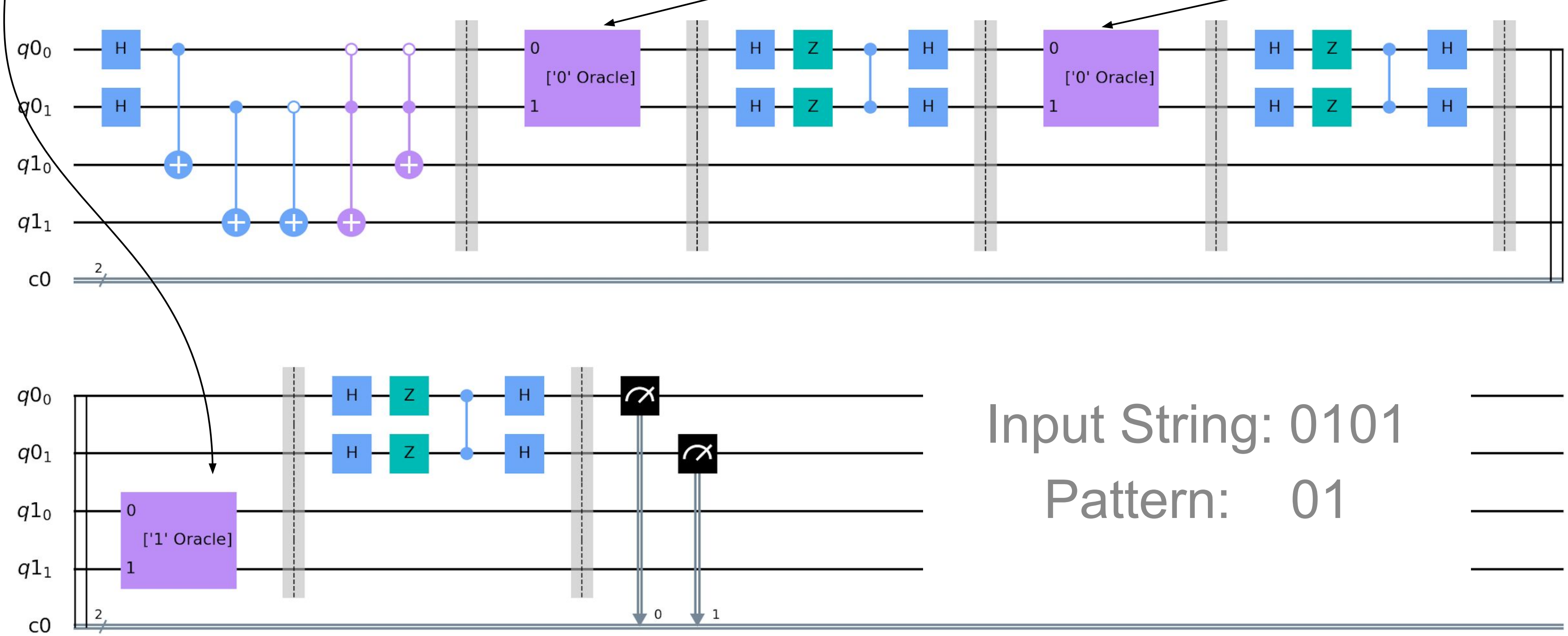
Notice: No ‘1’ Oracle in this run

“Many Oracles”, Gate Diffuser

```
Oracle for '1'[1]
[[ 1.  0.  0.  0.]
 [ 0. -1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

```
Oracle for '0'[0]
[[-1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

```
Oracle for '0'[2]
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0. -1.  0.]
 [ 0.  0.  0.  1.]]
```



Input String: 0101

Pattern: 01

Testing

Case: The characters in the pattern occur only in the matches, not in any other part of the input.

Example:

Input : bcatdfgh

Pattern: cat

Theoretical Match Percentage:

25%

** Tests included only patterns that existed in the input string

Latin Alphabet, **Single-Character** Patterns		
	"Many Oracles"	"Single Oracles"
Gate Diffuser	Fail: 23.33 % Match: 76.67 %	Fail: 26.67 % Match: 73.33 %
Matrix Diffuser	Fail: 26.67 % Match: 73.33 %	Fail: 15.83 % Match: 74.17 %

Latin Alphabet, **Multiple-Character** Patterns		
	"Many Oracles"	"Single Oracles"
Gate Diffuser	Fail: 65.45 % Match: 34.55 %	Fail: 71.36 % Match: 28.64 %
Matrix Diffuser	Fail: 70.00 % Match: 30.00 %	Fail: 69.55 % Match: 30.45 %

Open Problems

- Unclear if diffuser should be applied to all registers
- Description of oracle by authors as transposition operator using Gray codes
- Detecting patterns in alphabet consisting of only 0s and 1s
- Implementing exact pattern match detection